

APIs TO EXTRACT INFORMATION FROM AN EXISTING WEB RADIO APPLICATION

ANSELM MATHIAS, SAJID M. SHEIKH, ANNAH M. JEFFREY & SHEDDEN MASUPE

Department of Electrical Engineering, Faculty of Engineering, University of Botswana, Gaborone, Botswana

ABSTRACT

Web radio is a broadcast of an audio stream over the internet. In this day of computerization, just digitizing a task is no longer enough as automation is the key principle currently required to effectively use any resource available. Most radio applications do not provide integration facilities for micro services such as SMS or Voice. This paper presents APIs that were developed to extract information from an existing web radio application, in order to allow any developer of micro services to have the possibility to easily use the Web Radio. The web radio management known as Airtime was chosen to provide such integration capabilities with the use of RESTful APIs. Airtime is an Open Source Application which allows the control of the Web Radio. By reverse engineering the PostgreSQL database that supports the Airtime application, suitable functions were designed and then developed to meet the objectives of extracting information. The API functions created were show/search, show/details, track/search, track/details, stats/server, stats/icecast, playlist/search and playlist/details. The APIs perform searching for specific shows based on information provided, search for media that is available on the media server (mostly interviews, and podcasts) and provide statistical information such as application status from the servers. With the assistance of the Restler (Luracast) framework, Object Oriented PHP code and SQL, scripts were successfully written and tested to provide the required functionalities.

KEYWORDS: Airtime, API, Open Source, PHP, Radio, Restler, SQL, Web Radio

List of Abbreviations

AMF	:	Action Message Format
API	:	Application Programming Interface
ASCII	:	American Standard Code for Information Interchange
CSV	:	Comma Separated Values
HTML	:	Hypertext Transfer Protocol
HTTP	:	Hyper Text Transfer Protocol
JSON	:	Javascript Object Notation
MP3	:	MPEG-1 or MPEG-2 Audio Layer III
MSMQ	:	Microsoft Message Queuing
REST	:	Representational State Transfer
SaaS	:	Software as a Service
SMS	:	Short Messaging Service
SMTP	:	Simple Mail Transfer Protocol
SOAP	:	Simple Object Access Protocol

SQL	:	Sequential Query Language
TCP	:	Transmission Control Protocol
URI	:	Universal Resource Indicator
URL	:	Universal Resource Locator
W3C	:	World Wide Web Consortium
XML	:	eXtensible Markup Language

INTRODUCTION

Web Radio brings traditional radio to the digital era through access of Radio over the Internet. As technology evolves the need to do tasks manually has reduced considerably. This is achieved by promoting automation, either by use of scripts that run once a specific event has occurred, or developing multiple open ended Application Programming Interfaces (APIs), that do a variety of tasks which are integrated into a large range of products. Alternatively, by sharing code (open source-architecture), developers are given an opportunity to add more functionalities to an existing product instead to spending time accomplishing a task that is already done by someone else.

An API is a set of programming instructions and standards for accessing a Web-based software application or Web tool. A software company releases its API to the public so that other software developers can design products that are powered by its service. For example, Google uses APIs for showing weather reports, website visit statistics and many others. APIs follow the Software as a Service (SaaS) methodology, so that software developers do not have to start from scratch every time they write a program. For software developers with a C or C++ background, an API is similar to a function call [1].

An API is a software-to-software interface and not a user interface. The API itself is abstract, in that it specifies an interface and does not get involved with the implementation details. The software that provides the functionality described by an API is said to be an implementation of the API. With APIs, applications can interconnect, hence when operating and running, they are invisible to the users. These could be additional functions that are like frameworks to be used by future programmers to build onto, or to create added functionality [2,3]. The main objective of this research project was to develop APIs allowing any developer of micro service to have the possibility to use and control an existing webradio system. The micro service generates content and the APIs that will be created will be used ontop of the existing webradio management system. Theweb APIs should be useable from a php project to interact with a web radio system.

After considerable amount of research, an open source application called “Airtime” was selected to provide radio streaming/broadcasting services. However, the Airtime application did not have any intergration or automation facilities that could be used by Third party and micro service providers. The project also aimed at developing codes/APIs for achieving this functionality. Different API Architectures were scrutinised to find a suitable design that could be used in the development of the APIs for the Airtime software. Implementation of the Architecture, the design of the APIs and testing of the APIs were tasks that were acomplished for this project.

This project provided a product that can be used by and distributed with the application. The developed APIs would have the following functionalities;

- Search specific shows based on information provided.

- Be able to search media that is available on the media server (mostly interviews, and podcasts).
- Provide statistical information such as application status from the servers.

API FRAMEWORK

A framework is a universal reusable software platform to develop software or APIs. These include support programs, compilers, codes, libraries, APIs and tool sets. This section briefly presents the main web application framework that were studied, learned and used to code the APIs needed to provide the required functionalities.

HTTP Methods

The HyperText Transfer Protocol (HTTP) was designed and developed by the Internet Engineering Task Force and the World Wide Web Consortium (W3C) in 1999. This was to enable communication between clients and servers. HTTP was designed to use verbs (methods) to achieve a specific task or manipulate a specific resource. Some of these verbs are GET, POST, PUT and DELETE[4].

The actions performed by these verbs are based on their name, where GET requests a resource from the server and POST submits information to the server. The line below is an example of GET that assigns "variable 1" the value of "value1", and "variable 2" the value of "value2".

```
/test/form.asp?variable1=value1&variable2=value2
```

The same thing can be achieved using POST as shown in the snippet code below

```
POST /test/form.asp HTTP/1.1
```

```
Host: localhost.com
```

```
variable1=value1&variable2=value2
```

There are many formats in which the system can give an output such as XML, JSON, plain text and CSV. Thus, it becomes very crucial to understand and select an appropriate output format. It should be noted that plain text and CSV are not used in web-oriented architectures.

XML

Extensible Markup Language (XML) is a specific form of writing text, which is defined by the World Wide Web Consortium (W3C). It was designed to be easily implemented and used with HTML; it was also designed to be both human and machine-readable. XML has been used as the core language in communication of different protocols. XML uses tags - however, these tags are not predefined. The tag is defined by either a person or the automated machine. A sample of how XML looks is shown in Figure 1 [5].

```
<?xml version="1.0"?>
<response>
  <bmi>31.77</bmi>
  <message>Obesity</message>
  <metric>
    <height>162.6 centimeter</height>
    <weight>84 kilograms</weight>
  </metric>
  <imperial>
    <height>5 feet 4 inches</height>
    <weight>185.19 pounds</weight>
  </imperial>
</response>
```

Figure 1: Sample of XML Output [5]

JSON

JavaScript Object Notation (JSON) is a text format that is designed to be human readable and is used for interchange of data. JSON is derived from and supported within Javascript, whereas XML requires libraries to retrieve additional data. JSON is also language independent and was designed to be minimal, portable, and textual. Figure 2 represents a sample output in the JSON format.

```
{
  "bmi": 31.77,
  "message": "Obesity",
  "metric": {
    "height": "162.6 centimeter",
    "weight": "84 kilograms"
  },
  "imperial": {
    "height": "5 feet 4 inches",
    "weight": "185.19 pounds"
  }
}
```

Figure 2: Sample of JSON Output [6]

Through implementation and case studies, it is realized that JSON is considerably faster as it uses less resources, is easier to read and interpret by a human and is simpler to use than XML. It is crucial to understand that the APIs sole purpose is to manipulate data, where JSON fits more naturally as it is data-oriented in design [7]. JSON format was therefore used to display the output in this project [6].

SOAP-Based Web Service

Simple Object Access Protocol (SOAP) is a lightweight protocol intended for exchanging structured information in a decentralized, distributed environment. It makes use of XML technologies [4], by mostly relying on the application layer, and can be used over any transport protocol such as TCP, HTTP, SMTP, or even MSMQ. Developed by a group of vendors such as Microsoft, IBM, and Lotus, SOAP became a W3C Recommendation on 24th June 2003 [8, 9].

At the core, SOAP defines a way to move XML messages between two systems [10]. The SOAP message consists of a header element made up of header information, a body element that contains call and response information, and a fault element containing errors and status information. The actual SOAP message is shown in Figure 3.

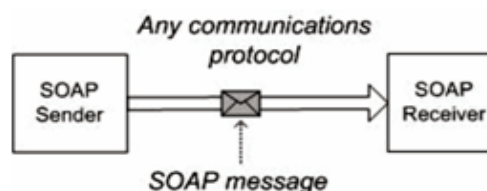


Figure 3: Simple SOAP Messaging [8]

In SOAP, XML is used to provide an output that is acquired from any scripts. SOAP provides flexibility such that other protocols can be stacked over the HTTP, which is used as the transportation protocol [11]. The above advantage can also pose as a security risk, as the SOAP standard also does not have any in-built security facilities. SOAP also has other problems, in terms that the service is only one-way (from client to server), and that it is slower than other technologies. [12]

RESTful Web Service

Representational State Transfer (REST) is a style of software architecture for distributed hyper media systems such as the World Wide Web. REST provides a set of architectural constraints that, when applied as a whole, emphasizes

scalability of component interactions. A general construct is shown in Figure 4. Roy Thomas Fielding coined the term “Representational State Transfer”, in his Doctorate dissertation entitled “Architectural Styles and the Design of Network-based Software Architectures”[13].

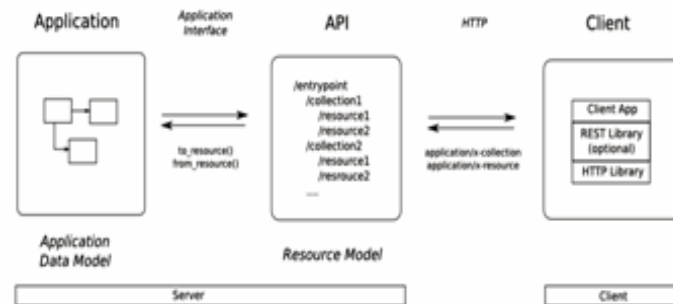


Figure 4: REST Relationship Diagram [13]

An alternative way to distinguish and define REST would be to say that it is a stateless server, where each request from a client contains all the information necessary to service the request, which clearly separates the clients and servers. REST clients can cache responses if allowed and there is a direct interface between clients and server. The client can then request code from the server and execute it on demand. REST has a loosely typed architecture, but uses nouns and verbs to promote flexibility and usability. REST is not restricted to parsing out only XML, it also uses less bandwidth and provides error checking[14,15].

In order for REST APIs to work, it should be able to identify the resource, e.g. by using the Universal Resource Indicator (URI), and also be able to manipulate the resource through representation (i.e. modify or delete), using the HTTP request of GET, POST, PUT, and DELETE. REST’s own flexibility can be a problem since there is no common accepted standard for REST, hence interoperability between large systems might be a problem. [16, 17,18]. By using the above architectures, it clearly demonstrates how the system would get the data, and how it would respond. Having understood the advantages and disadvantages of each type of entity, the REST architecture was selected for this project, as it would perfectly fit the scalability needs of the application.

Restler Framework

There are many Frameworks that are designed for various reasons and in different languages, such as Tonic, Restler, Slim, FRAPI, Zend Framework just to name a few[19]. From these many options, the Restler framework was selected as this web-API framework provides easy deployment of the REST architecture. Restler is prebuilt using different applications to provide an inbuilt solution thus avoiding the re-writing of the code. It is very simple to use if object-oriented programming of PHP is already understood. Restler also supports different formats like JSON, XML, yaml, amf, plist.

Emerginov Platform

This is an open-source solution developed by Orange Labs, which provides the opportunity to build application with the use of service such as SMS and Voice. One such platform is located locally in Botswana, for the encouragement of local industries to use it, create solution, and nurture ideas in fields such as m-health and agriculture. This platform was used to demonstrate the capabilities of the API [20, 21].

GUIDELINES

In designing API's there are no fixed rules, however, there are the recommended practices that have been

perfected over the years of development by professionals. The guidelines below have been used in the code design and implementation [22]

- **Structure:** Normally APIs follow the standard design procedure shown in the Figure 5.



Figure 5: Anatomy of API [22]

- **Versioning:** It is very important to provide versioning as it helps in large scale development without hindering older users. For example, Facebook versions such as "?v=1.0"
- **Status Codes:** Use of HTTP status codes to relevant standard-based codes for errors, e.g. Google uses 200, 201, 304, 400, 401, 403, 404, 409, 410, 500. Information that is provided by the server should be as descriptive as possible in order to relay any problems that have occurred which needs to be rectified.

For example:

```

{
  "status": 409
  "property": "name",
  "message": "A directory named ' Avengers' already exists",
  "developerMessage": "a directory named 'Avenger' already exists. If you have a stale local cache, please expire it now."
  "moreInfo": "www.check it out.com/errors/31337"
}
  
```

For the code designed, the inbuilt Restler error codes were used.

- **Authentications – Sessions** are to be avoided whenever possible and provide authentication for every request if necessary. Existing protocol such as Oauth 1.0a, Oauth2 or SSL are to be used. In this project, the authentication is done by the Emerginov platform.

METHODOLOGY

Implementation Architecture

Due to the complexity of building an open-source application and the constant changes associated with it, the APIs were to be built by manipulating the PostgreSQL Database on which the application was built upon. Other factors that also affected this decision were the speed of computation and reducing points of failure. The envisioned implementation is such that the APIs are to be built within the Airtime application, and should be available only to the

developers that are authenticated by the Emerginov platform. The listeners, being unaware of such complexities, would only be able to access the broadcast that is given out from Icecast. This is demonstrated in the Figure 6.

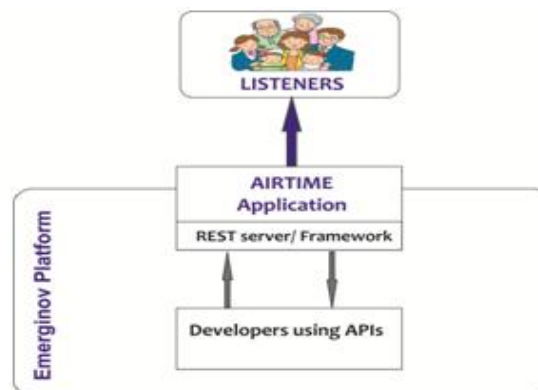


Figure 6: Design of Proposed Implementation

Proposed API Functionalities

The APIs that needed to be developed to extract data are meant to provide as much information from the Airtime Application's features that already exist. Below are functions that have to be carried out by the APIs to be developed:

- **Search Shows:** This will allow developers to acquire information pertaining to show line-ups. This contains information such as; which shows are when, who is the host etc. By doing a search, a developer can get additional information using the "details" option.
- **Search Tracks:** This option would help search for tracks based on the criteria provided, and even provide more present if requested.
- **List Radio Statistics and Media Information:** This would allow the developers to access the meta data of tracks held in the database, and the statistics of the server, that can be used for maintenance gathered by the application.

Detailed Description of the Services and Existing Database Model

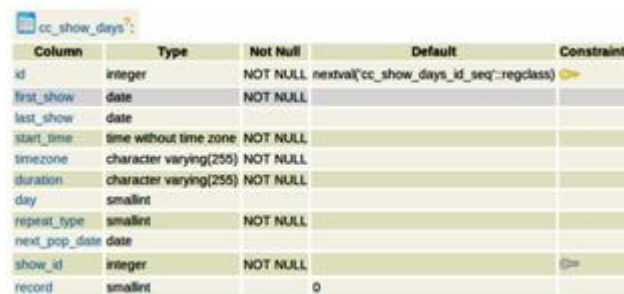
Since the manipulation of the application is done through the database, it is important to understand the tables and the information each field holds. However, a small extract is shown below and complete relational database is shown in figure 7.

- **Shows:** Currently when a show is updated, its details are stored in the table labelled "cc_show" shown in Figure 8. This mostly contains name of the show and few other details dealing with aesthetics of how the show will look in the calendar. It also has options for authenticating users who would join into the live stream, either using the authentication already existing in the Airtime user database or using a predetermined username and password.

Column	Type	Not Null	Default	Constraints
id	integer	NOT NULL	nextval('cc_show_id_seq::regclass')	
name	character varying(255)	NOT NULL	''::character varying	
url	character varying(255)		''::character varying	
genre	character varying(255)		''::character varying	
description	character varying(512)			
color	character varying(6)			
background_color	character varying(6)			
live_stream_using_airtime_auth	boolean		false	
live_stream_using_custom_auth	boolean		false	
live_stream_user	character varying(255)			
live_stream_pass	character varying(255)			

Figure 8: Table cc_show

The information about when the show is to be started is stored in another table called "cc_show_days" (Figure 9). This table further shows the starting date and time, and the duration of the show and the date of when the show ends, assuming that the shows are repeated. In addition, "day" represents the day of the week, which starts Sunday as "0".



Column	Type	Not Null	Default	Constraints
id	integer	NOT NULL	nextval('cc_show_days_id_seq'::regclass)	
first_show	date	NOT NULL		
last_show	date			
start_time	time without time zone	NOT NULL		
timezone	character varying(255)	NOT NULL		
duration	character varying(255)	NOT NULL		
day	smallint			
repeat_type	smallint	NOT NULL		
next_pop_date	date			
show_id	integer	NOT NULL		
record	smallint		0	

Figure 9: Table cc_show_days

Relationship Diagram: In order to manipulate the data on the server side, it is necessary to understand the relationship between the tables themselves. This means that when data is changed in one table, the related table should also be changed using the SQL "JOIN" functions, so as to not break the database. In Figure 7 the lines show how specific fields are connected to other fields in the tables.

Proposed Error Codes

The error codes are important as they are a means to provide insight into the inner workings of a code without going through the code itself, in case a problem arises from the use of the code. The most standardised error messages were written for the HTTP, from which these error codes were derived [3].

Table 1 below describes all the errors codes and the description associated with them, which were used in the development of the APIs. These are few of the error codes that are built into the Restler framework, that are going to be used in the APIs.

Table 1: Proposed Use of Errors

Error Code	Error Text	Description
200	Ok	This is when the request has been successfully carried out.
201	Created	Resource has been successfully modified.
400	Bad Request	The request cannot be fulfilled due to bad syntax.
404	Not Found	The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible.
500	Internal Server Error	This is when either a connection or query has failed to get the required information.

RESULTS

This section presents the data manipulation, using the specified HTTP method call used for each service and the API coding. Each service is broken down into a brief description of what the service is supposed to do, followed by the parameters that are required to process the request. The parameters which are closely associated with the table field names are manipulated and then stored in the database. These parameters can be either optional or required. The HTTP method GET is mostly used to acquire data, and POST is used when inserting data as well as dealing with anything other than

American Standard Code for Information Interchange (ASCII) e.g. media files such as mp3's. The output results contain the information that will be obtained by the use of the specific API. This information is summarized and placed in Table 2 presented at the end of this paper.

Show Data API

An example is provided here for the "show/details" API. In order to use this API the show ID must be provided. When this API is called, the name of the show, the genre, the description of the show, when the show starts, and when it ends is displayed. If it is repeating, it will show all the entries as well as the expected outputs.

API Coding

In order to produce the required results, object-oriented programming of PHP had to be used by the use of classes. The sample extracts of "index.php" are shown below. This page is accessed through following "http://localhost/apiv1/index.php" on the local machine after deployment of Airtime. The "index.php" is the main page that any site starts. As soon as the web server calls on the "index.php" file, the Restler framework shown by the snippet below was loaded.

```
require_once '../vendor/restler.php';

useLuracast\Restler\Restler;
```

A new instance of the object Restler is created, and then parses the data that is required for the class function to be acted upon as shown below. This is how Restler is controlled and works. The default format is XML, so Restler has to be manually set to JSON format

```
$r = newRestler();

$r->setSupportedFormats('JsonFormat');

$r->addAPIClass('playlist');
```

The API then goes to acquire the credential stored in the "/etc/airtime/airtime.conf" file, which is required in connection to the PostgreSQL database and is stored in an array called "airtime_config" as shown below.

```
$airtime_config = parse_ini_file("/etc/airtime/airtime.conf") ;
```

The snippet below shows when a connection called "dbconn" is established with the credentials acquired.

```
$dbconn= pg_connect("host=$airtime_config[host]
dbname=$airtime_config[dbname] user=$airtime_config[dbuser]
password=$airtime_config[dbpass]");
```

Error checking is done on every stage; the code snippet below is a condition that gets called incase the connection to the database fails. This is very crucial as any tasks and functions that are to follow will not work, hence the web server is forced to "exit" before parsing any other lines.

```
if (!$dbconn) { // error if the connection fails

echo "An error occurred connecting to Database. \n";

exit;

}
```

The APIs are be characterized by the notion of data entering (insertion) the system, or data leaving (extraction) the system.

APIs Created for Extracting Data

There is a class called "playlist" and a function called "search" in it that has four variable called "playlist_name", "playlist_description", "limit", and "page". The latter two variables are pre-set, while the former two are empty as shown in the snippet below.

```
class playlist {
    function search ($playlist_name = '*', $playlist_description = '*', $limit='25',$page='1') {
```

The "limit" and "page" number variables are used in breaking up the results into pages, this process is called pagination. This is done to prevent the over flow of search results, that either leads to unpleasant use of the application or the bottlenecking of the system resources. An example of how pagination is used is shown below, the variables are then used in the SQL statement.

```
if ($page > 0){
    $page = $page - 1;
    $page = $limit * $page;
}
```

Most variables that are provided are firstly converted to the upper cases, shown in the code below. This is to ease the process of searching, as well as avoiding errors.

```
$playlist_name = strtoupper($playlist_name);
$playlist_description = strtoupper($playlist_description);
```

Once the values have been assigned to the specific function, the function then calls in the global connection variable to see if the connection to the database is successful. The defined values are then used as part of an SQL search from the database, whose results are stored in a variable labelled "result". The SQL query is designed to use wildcards in the beginning and the end of a string variable. The process below shows an example of such a process.

```
$result = pg_query($dbconn, "SELECT id, name, description FROM cc_playlist where ('$playlist_name' = '*' or upper(name) like '%$playlist_name%') and ('$playlist_description' = '*' or upper (description) like '%$playlist_description%') order by id asc limit $limit offset $page" )
```

The variable result is one big array, and hence to break it down, a loop had to be designed to split the information line by line. This is then broken down into "rows" and stored in a new array, which is assigned to "tempObject". The "tempObject" is assigned to "resultant" and this process of assignment is repeated till there are no more results. This is shown by the snippet below.

```
$number_of_rows = pg_num_rows($result) ;
if ($number_of_rows > 0) {
    while ($row = pg_fetch_array($result)){
        $tempObject = new Stdclass () ;
```

```

$tempObject->Id = $row[0];

$tempObject->Name= $row[1];

$tempObject->Description= $row[2];

$resultant [ ] = $tempObject;

}

return $resultant ;

}

```

“resultant” is then returned from the function back to the main page, by doing so it escapes the function and clears up the “result” variable. Restler then pushes the output to the client using the syntax shown below.

```
$r->handle();
```

After this, all the information is removed from memory and the connections are closed. This is done for security reasons and to free up system resources.

```

pg_free_result($result);

pg_close($dbconn);

```

This process is repeated for every class and every function in the same way when called upon. The above function uses the GET request, to accomplish the task of acquiring information from the system.

Testing

Testing is an important phase in the development of any product. Below are different tests carried out to find the durability of the developed code and possibly rectify where it has its shortfalls. The input value for playlist details is the playlist ID (which is an integer). So, if a string is placed as shown in Figure 10, an exception is thrown as shown in Figure 11, which shows an error "400". By referencing it to the design, it means that a bad syntax has been used.



Figure 10: Variable Testing

```

{
  "error": {
    "code": 400,
    "message": "Bad Request: not valid input"
  }
}

```

Figure 11: Error, Bad Syntax

It is rare to find an internal error, but it has to be catered for. Table 3 summarizes the values of the results that were obtained under testing of the APIs created.

Table 2: Summarized Outcome of Testing

Class/ Function	Tested Variable /Type	Input Type	Input Value	Expected Response	Response Acquired
Show/ search	showname /string	String	new	Pass	pass
	Genre/ String	String	Rap	Pass	pass
	DJ_name /String	String	Stig	Pass	pass
	time_heard /Date	Date	10-01-2013	Pass	pass
	time_heard /Date	String	10th Nov	Fail	fail
	Limit / integer	Integer	20	Pass	pass
	Limit / integer	String	twenty	Fail	fail
	Page /integer	Integer	10	Pass	pass
	Page /integer	String	Ten	Fail	fail
Show/ details	Showid /integer	Integer	10	Pass	pass
	Showid /integer	String	Ten	Fail	fail
	Limit / integer	Integer	20	Pass	pass
	Limit / integer	Integer	twenty	Fail	fail
Track/ search	trackname /String	String	Me	Pass	Pass
	genre/String	String	Rap	Pass	pass
	artist_name /String	String	Stick	Pass	Pass
	album_title/String	String	Dance	Pass	pass
	limit/integer	Integer	10	Pass	pass
	limit/integer	Integer	Ten	Fail	fail
	page/integer	Integer	20	Pass	pass
	page/integer	Integer	twenty	Fail	fail
Track/ details	Trackid/integer	Integer	20	Pass	pass
	Trackid/integer	String	twenty	Fail	fail
Playlist/ search	playlist_name /String	String	mix	Pass	pass
	playlist_description /String	String	All the house	Pass	pass
	Limit / integer	Integer	20	Pass	pass
	Limit / integer	String	twenty	Fail	fail
	Page /integer	Integer	10	Pass	pass
	Page /integer	String	Ten	Fail	fail
Playlist/ details	Playlist_id /integer	Integer	5	Pass	pass
	Playlist_id /integer	String	New	Fail	Fail
Stats/ icecast	NULL	NULL		Pass	pass
Stats/ server	NULL	NULL		Pass	pass

DISCUSSIONS

Creating a solution for a live (already in production) open source application is a very daunting task as it always remains a "cat and mouse game" between the independent developer (who are adding new functionalities to the product), the core developers (those who develop the application) and the bug testers. Hence, it is very important to build any third party application quickly and pass it on to the core developers who would then integrate it into the next update. In order to do this, a team of developers are required especially to achieve this task. The development of the APIs had started on Airtime version 2.2, and most functions were built for this. However, version 2.3 was then released on which some function failed to work completely, and as such the API development had to restart in relation to a few functions that were updated in the new version.

Other problems encountered were the lack of documentation associated with the technical design of the application. As it is open source, most developers add functionality without providing the technical documentation of the design procedure as it is not a key requirement. A lot of time was spent to reverse-engineer the application to find out what their functions are, where and how they are used, the security, and the validations associated with it. This was realised

when working with the Airtime application, as the database documentation was non-existent. Working with Open Source applications such as the one used in this project without documentation on coding can make API development a challenge.

CONCLUSIONS

The APIs developed are completely open source. The APIs provide a range of functions that are available on the application, but are not available to the developers. The developed APIs will provide external developers with the opportunity in accessing the application without having to physically login, but having valid platform credentials.

The many APIs coded have various functions, where an API called "show/search" was developed to search any show in the database, based on the show name, the genre, the time the show was heard, and even the DJ name. More details could further be acquired by using the "show/details" API. Another API called "track/search" would search through the database by comparing it to the information provided. This information could then be used to download the file. The "stats/icecast" and "stats/serverAPIs that were built would list the server statistics for remote monitoring of the server, from servers such as Icecast when called upon.

The HTTP method GET request was mostly used when requesting information from the server and the POST request is called to manipulate information in the radio application such as uploading of files and editing of data.

ACKNOWLEDGEMENTS

The Authors will like to thank Morgan Richomme and Arnaud Morin from Orange Labs for their guidance on the API coding and the use of the Emerginov Platform. The Authors will also like to thank the community namely from "Stackoverflow" and "sourcefabric" in assisting when issues with the code were encountered.

REFERENCES

1. Software & Information Industry Association. (2001, February). Software as a Service: Strategic Backgrounder. Retrieved February 28, 2013, from Software & Information Industry Association: <http://www.siiia.net/estore/pubs/SSB-01.pdf>
2. How to Leverage an API for Conferencing. (n.d.). Retrieved April 26, 2013, from How stuff works: <http://money.howstuffworks.com/business-communications/how-to-leverage-an-api-for-conferencing3.htm>
3. Rouse, M. (2005, September). SOAP (Simple Object Access Protocol). Retrieved April 26, 2013, from Techtarget: <http://searchsoa.techtarget.com/definition/SOAP>
4. R. F. (1999). Hypertext Transfer Protocol -- HTTP/1.1. Retrieved April 12, 2013, from world Wide web consortium: <http://www.w3.org/Protocols/rfc2616/rfc2616.html>
5. Extensible Markup Language (XML) 1.0 (Fifth Edition). (2008, November 26). Retrieved April 4, 2013, from world wide web consortium: <http://www.w3.org/TR/REC-xml/>
6. Nurzhan Nurseitov, M. P. Comparison of JSON and XML Data Interchange Formats: A Case Study.
7. drrwebber. (2013, April 26). Analysis of JSON use cases compared to XML. Retrieved May 12, 2013, from Oracle blogs: https://blogs.oracle.com/xmlorb/entry/analysis_of_json_use_cases
8. W3C Recommendations (2007, April 27). SOAP Version 1.2 Part 1: Messaging Framework (Second Edition). Retrieved January 23, 2013, from World Wide Web Consortium: <http://www.w3.org/TR/soap12-part1/#intro>

9. Kyrnin, J. (n.d.). web design /HTML . Retrieved December 29, 2012, from About.com:
<http://webdesign.about.com/od/soap/a/what-is-xml-soap.htm>
10. SOAP Introduction. (n.d.). Retrieved march 12, 2013, from w3schools.com:
http://www.w3schools.com/soap/soap_intro.asp
11. Skonnard, A. (2003, march). Understanding SOAP. Retrieved January 18, 2013, from MSDN:
<http://msdn.microsoft.com/en-us/library/ms995800.aspx>
12. The Advantages and Disadvantages of Using SOAP Messages. (n.d.). Retrieved February 23, 2013, from xyzws:
http://www.xyzws.com/scdjws/studyguide/soap_chapter8.html
13. Fielding, R. T. (2000). Architectural Styles and the Design of Network-based Software Architectures. Retrieved October 21, 2012, from http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
14. Olson, M. (2002, july 03). The Python Web services developer: Messaging technologies compared. Retrieved march 15, 2013, from IBM Developer Works: <http://www.ibm.com/developerworks/library/ws-pyth9/>
15. Elkstein, D. M. (n.d.). Rest Server Responses. Retrieved January 12, 2012, from Learn REST: A Tutorial:
<http://rest.elkstein.org/2008/02/rest-server-responses.html>
16. Oracle. (n.d.). Web Services: REST vs. SOAP. Retrieved February 18, 2013, from Milan's blog by Oracle:
https://blogs.oracle.com/milan/entry/web_services_rest_vs_soap
17. SOAP vs. REST. (2010, January 15). Retrieved March 12, 2013, from spf13: <http://spf13.com/post/soap-vs-rest>
18. Huppo. (2011, june). Compare and contrast REST and SOAP web services. Retrieved march 14, 2013, from stackoverflow: <http://stackoverflow.com/questions/10975863/compare-and-contrast-rest-and-soap-web-services>
19. Lane, K. (2011, Septmeber 23). Short List of RESTful API Frameworks for PHP. Retrieved March 14, 2013, from Programmableweb: <http://blog.programmableweb.com/2011/09/23/short-list-of-restful-api-frameworks-for-php/>
20. Orange. (2012, December 04). Emerginov, An innovative solution for mobile services development in Africa. Retrieved February 19, 2013, from Orange news: <http://www.orange.com/en/news/2012/novembre/Emerginov-an-innovative-solution-for-mobile-services-development-in-Africa>
21. Orange. (n.d.). About. Retrieved February 19, 2013, from Emerginov: <http://www.emerginov.org/about.php>
22. Jansen, G. (2011). The Job of the API Designer. Retrieved January 19, 2013, from Restful API design: <https://restful-api-design.readthedocs.org/en/latest/scope.html>

APPENDICES

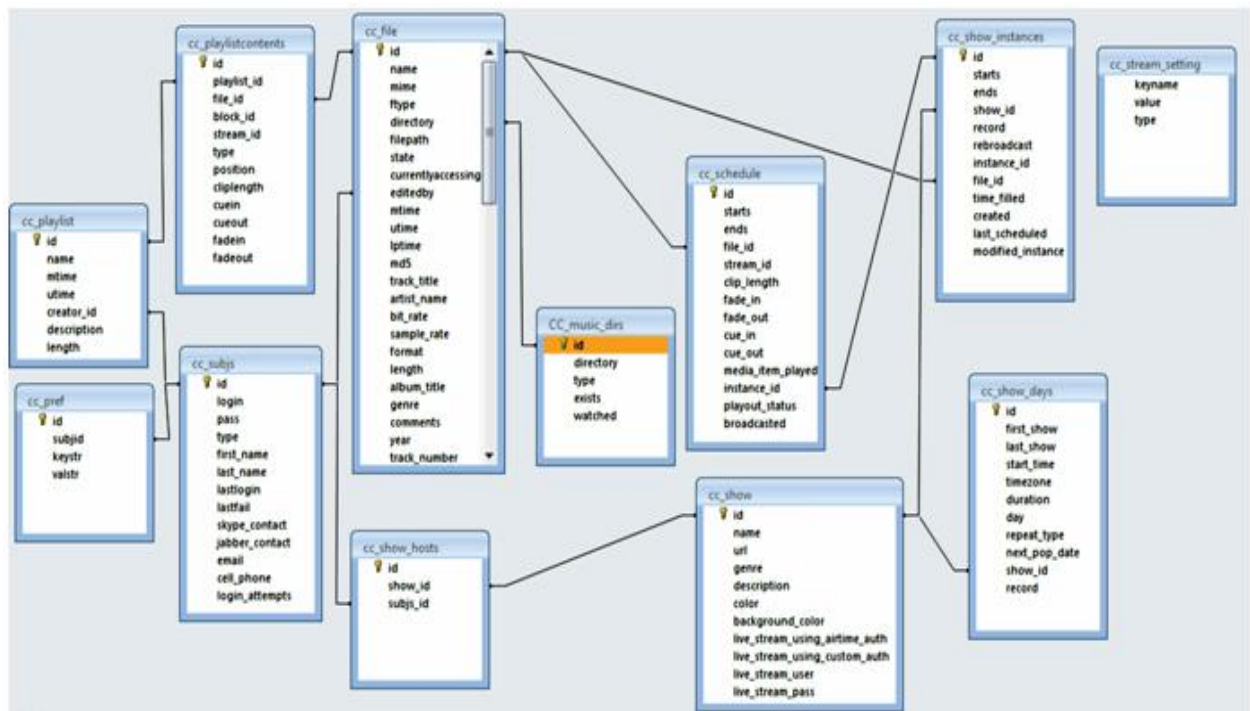


Figure 7: Relationships of Table in the Database

Table 3: Inputs, Methods and Outputs Associated to an API Function

API Name	Description	Parameters	Parameters Type	HTTP Method	Output Results Contents
Show/search	This can list all the show in the database, or can be refined by any field as criteria	name - list the name of the show genre - list the genre of the show time_heard - to refine search parameter using YYYY-MM-DD HH:MM DJ_name - name of the host	<i>Optional</i>	<i>GET</i>	Show_id, show_name, Show_genre, show_description, show_starts, show_ends, DJ_name.
Show/details	this should list all details available about a specific show	show_id - list the name of the show	<i>Required</i>	<i>GET</i>	Show _ name, show_genre , show_description , show_start_time (yyyy-mm-ddhh:mm:ss), show_duration, Host_name
Track/search	This can list all the tracks in the database, the can be refined by any field as a criteria, This is a more generalized search, that uses wildcards.	track_title - list the tracks associated with that name artist_name - list the tracks associated with an artist album_title - list the tracks associated with the specified album name Genre - list tracks with a specific genre	<i>Optional</i>	<i>GET</i>	Track_id , track_name,, artist_name, album_title, Genre

Table 3: Contd.,

Track/details	this will give out all the information available (the metadata) on the file that is associated to the track id.	Track_id	<i>Required</i>	<i>GET</i>	Id , track_title, artist_name, album_title, Genre, disc_number, mood, label, track_number, copyright, length
Stats/server	This is where all the statistics are available for analysis, from media-monitor	NONE	<i>Required</i>	<i>GET</i>	Current_listeners, Peak_listeners, current_song, content_type, Airtime_Version, Playout_Engine_Running_Seconds, Playout_Engine_Mem_Perc, Playout_Engine_Cpu_Perc, Liquidsoap_Running_Seconds, Liquidsoap_Mem_Perc, Liquidsoap_Cpu_Perc, Media_Monitor_Running_Seconds, Media_Monitor_Mem_Perc , Media_Monitor_Cpu_Perc, Rabbitmq_Running_Seconds, Rabbitmq_Mem_Perc, Rabbitmq_Cpu_Perc
Stats/icecast	This is where all the statistics are available for analysis, from icecast	NONE	<i>Required</i>	<i>GET</i>	Title, description, content_type, mount_start, bitrates Listeners, most_listeners, genre, url
Playlist/search	This method assists the developer to search and list playlists (could be used with keywords or wild cards ie *)	Playlist_name playlist_description	<i>Optional</i>	<i>GET</i>	Playlist_name, playlist_descriptio, playlist_id
Playlist/details	This method gives more details about the contents in a playlist, this will be done by exploiting the table_cc_playlist and table_cc_playlistcontents, table_cc_file	Playlist_ID	<i>Required</i>	<i>GET</i>	Playlist_name, Playlist_duration, files_in_playlist, playlist_id